우리동네단골 포팅 메뉴얼

1. 개요

1.1. 프로젝트 개요

- 프로젝트 명
    - 우리동네단골
- 프로젝트 소개
    - 우리동네GS 재구매 추천 알고리즘 적용 및 대시보드를 이용한 현황 파악
- 주요 기능
    1. 사용자 정보 수집을 위한 우리동네단골 App
    2. 사용자 정보(성별, 나이, 상품 조회, 장바구니, 구매)를 바탕으로 재구매 추천 알고리즘
    3. 사용자 정보 현황 파악을 위한 우리동네단골 대시보드

1.2. 프로젝트 사용 도구

- 이슈 관리
    - JIRA
- 형상 관리
    - Gitlab

- 커뮤니케이션
  - Notion
  - Mattermost
- 디자인
  - Figma
- DB 설계
  - ERD Cloud
- 동영상 편집
  - 모바비
  - vrew

1.3. 개발 환경

- 프론트엔드
  - Android Studio
  - VSCode
- 백엔드
  - IntelliJ IDEA
  - VSCode
  - PyCharm
- 인프라
  - gitbash
- DB
  - MySQL Workbench
  - SQLTools(VSCode extensions)
  - MongoDBCompass

1.4. 기술 스택

- 프론트엔드
  - Andriod Studio 2024.1.1
  - Dart 3.5.3
  - Flutter 3.24.3
  - Get 4.6.5
- 백엔드

  프로그래밍 언어 및 런타임

    - Java 17

  프레임워크

    - Spring Boot 3.3.5
        - Spring Data JPA
        - Spring Web
        - Spring Security
        - Spring Validation
        - Spring WebFlux (WebClient)

  보안

    - JWT (Json Web Token): `io.jsonwebtoken:jjwt-api 0.12.6`

  API 문서화

    - SpringDoc OpenAPI: `springdoc-openapi-starter-webmvc-ui 2.6.0`

  Firebase

    - Firebase Admin SDK: `firebase-admin 9.0.0`

  개발 도구

    - Spring DevTools
    - Lombok
- DB
    - Elasticsearch 7.17.0
    - MySQL 9.0.1
    - Redis 7.4.0
    - Cassandra 5.0.2
    - MongoDB 8.0.3
- 인프라
    - AWS EC2
    - AWS S3
    - Docker
    - Jenkins
    - NginX

1.5. 추가 기술 스택

- 추천 알고리즘 관련
    - FastAPI 0.115.0
    - numpy 1.24.3

- scikit-learn 1.24.3
- pandas 2.2.3
- scikit-learn
- mkl-service 2.4.0
- mkl-random 1.2.2
- mkl_fft 1.3.0 이상
- Python 3.9

1.6. 서버 목록

1. SSAFY EC2(t2.xlarge)
    a. 인스턴스 유형: EC2 ㅇt2.xlarge
    b. VCPU: 4
    c. RAM: 40GB
        - 기본 RAM: 16GB
        - Swap RAM: 24GB
    d. 스토리지: 300GB
    e. 컨테이너
        i. Nginx
            - 역할: Reverse Proxy
            - 포트: 80
        ii. Jenkins
            - 역할: CI/CD 파이프라인 관리
            - 포트: 9090
        iii. React 애플리케이션
            - 역할: 정적 파일 제공(대시보드)
            - 포트: 80
        iv. Spring 애플리케이션 1
            - 역할: client용 백엔드 API 서버
            - 포트: 8080
        v. Spring 애플리케이션 2
            - 역할: manager용 백엔드 API 서버
            - 포트: 8081
        vi. Mysql
            - 역할: 관계형 데이터베이스(회원 관리 및 기본 상품 데이터 등)

- 포트: 3306

vii. Elasticsearch
- 역할: 검색 엔진
- 포트: 9200

viii. Redis
- 역할: 캐시 관리(firebase token 저장)
- 포트: 6379

ix. MongoDB
- 역할: NoSQL 데이터베이스(알림 저장)
- 포트: 27017

2. EC2 인스턴스 1(t3.2xlarge)
   a. 인스턴스 유형: EC2 t3.2xlarge
   b. VCPU: 8
   c. RAM: 52GB
      - 기본 RAM: 32GB
      - Swap RAM: 20GB
   d. 스토리지: gp3 - 300GB
   e. 루트 디바이스 유형: EBS
   f. EBS 최적화: 활성화
   g. 컨테이너
      i. Nginx
         - 역할: Reverse Proxy
         - 포트: 80
      ii. Jenkins
         - 역할: CI/CD 파이프라인 관리
         - 포트: 9090
      iii. FastAPI 애플리케이션
         - 역할: 추가 API 서버(추천 시스템, 데이터 처리)
         - 포트: 8000
      iv. Spark Master, Worker
         - 역할: 분산 데이터 처리를 위한 Spark
         - Spark Master 컨테이너:
            - 포트: 7077, 8080
         - Spark Worker컨테이너:

        - 포트: 8081
    v. Cassandra 클러스터 (노드 3개)
        - 역할: 분산형 NoSQL 데이터베이스(6개월 이내 raw log)
        - 포트: 9042
3. EC2 인스턴스 2(t2.xlarge)
    a. 인스턴스 유형: EC2 t2.xlarge
    b. VCPU: 4
    c. RAM: 40GB
        - 기본 RAM: 16GB
        - Swap RAM: 24GB
    d. 스토리지: gp3 - 300GB
    e. 루트 디바이스 유형: EBS
    f. EBS 최적화: 활성화
    g. 컨테이너
        i. Spark Master, Worker
            - 역할: 분산 데이터 처리를 위한 Spark
            - Spark Master 컨테이너:
                - 포트: 7077, 8080
            - Spark Worker컨테이너:
                - 포트: 8081
        ii. MondgoDB
            - 역할: NoSQL 데이터베이스(6개월 지난 raw log 및 기업데이터 저장)
            - 포트: 27017

2. 빌드

2.1. 프론트엔드 빌드 방법

- **APP(Flutter)**

1. 버전
    a. Flutter 3.24.3
2. 라이브러리 설치
    a. pubspec.yaml 파일에 아래 설정 추가

```
name: odd
description: "A new Flutter project."
publish_to: "none" # Remove this line if you wish to publish to pub.dev
```

```
version: 1.0.0+1

environment:
  sdk: ^3.5.1

dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.8
  intl: ^0.17.0
  get: ^4.6.5
  get_storage: ^2.1.1
  flutter_dotenv: ^5.2.1
  http: ^0.13.6
  shared_preferences: ^2.0.13
  firebase_core: ^3.7.0
  firebase_analytics: ^11.3.4
  firebase_messaging: ^15.1.4
  flutter_local_notifications: ^17.2.3
  flutter_native_splash: ^2.4.2
  cached_network_image: ^3.2.3

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^4.0.0

flutter_native_splash:
  color: "#02D4EC"
  image: assets/images/odd_start.png
  android_12:
    image: assets/images/odd_start.png
  android: true

flutter:
  uses-material-design: true

  assets:
    - .env
    - assets/images/
    - assets/images/home/
    - assets/images/cart/
    - assets/images/order/
    - assets/images/notification/
    - assets/icons/
    - assets/icons/navbar/
```

```
  fonts:
    - family: LogoFont
      fonts:
        - asset: assets/fonts/Dunggeunmiso-B.ttf
          weight: 700
        - asset: assets/fonts/Dunggeunmiso-R.ttf
    - family: Freesentation
      fonts:
        - asset: assets/fonts/Freesentation-3Light.ttf
          weight: 300
        - asset: assets/fonts/Freesentation-4Regular.ttf
          weight: 400
        - asset: assets/fonts/Freesentation-5Medium.ttf
          weight: 500
        - asset: assets/fonts/Freesentation-6SemiBold.ttf
          weight: 600
        - asset: assets/fonts/Freesentation-7Bold.ttf
          weight: 700
        - asset: assets/fonts/Freesentation-8ExtraBold.ttf
          weight: 800
```

b. pubget 실행

```
flutter pub get
```

3. 빌드

a. apk 추출(안드로이드)

```
flutter build apk --release --target-platform=android-arm64
```

b. 빌드 파일 주소

```
\S11P31S105\frontend\app\odd\build\app\outputs\flutter-apk
```

c. 파일명 수정 후 배포

- **WEB(React)**

1. 버전

a. React 18.3.1

b. Nodejs 20.15.0

2. 라이브러리 설치

```
npm install
```

3. 빌드

```
npm run build
```

2.2. 백엔드 빌드 방법

1. 버전
    a. JAVA Open-JDK 17
    b. SpringBoot 3.3.5
    c. Gradle 8.10.2
2. 빌드

- **Spring-client**

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.3.5'
    id 'io.spring.dependency-management' version '1.1.6'
}

group = 'odd'
version = '0.0.1-SNAPSHOT'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'io.jsonwebtoken:jjwt-api:0.12.6'
    runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.12.6'
    runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.12.6'

    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-validatio
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation group: 'org.springdoc', name: 'springdoc-openapi-starte
    implementation 'io.netty:netty-all:4.1.78.Final'
    implementation 'org.springframework.boot:spring-boot-starter-data-elas
    implementation 'org.elasticsearch.client:elasticsearch-rest-high-level

    // Firebase Admin SDK
```

```
    implementation 'com.google.firebase:firebase-admin:9.0.0'

    // redis
    implementation 'org.springframework.boot:spring-boot-starter-data-redi
    implementation 'redis.clients:jedis'

    // mongodb
    implementation 'org.springframework.boot:spring-boot-starter-data-mong

    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'

    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.mysql:mysql-connector-j'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'

    // webclient
    implementation 'org.springframework.boot:spring-boot-starter-webflux'
    implementation 'io.netty:netty-all'

    // cassandra
    implementation 'org.springframework.boot:spring-boot-starter-data-cass
}

tasks.named('test') {
    useJUnitPlatform()
}
```

- **Spring-manager**

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.3.5'
    id 'io.spring.dependency-management' version '1.1.6'
}

group = 'odd'
version = '0.0.1-SNAPSHOT'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}

repositories {
    mavenCentral()
```

```
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-validatio

    // swagger
    implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.6.

    // lombok
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'

    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.mysql:mysql-connector-j'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'

    // webclient
    implementation 'org.springframework.boot:spring-boot-starter-webflux'
    implementation 'io.netty:netty-all'
}

tasks.named('test') {
    useJUnitPlatform()
}
```

```
./gradlew build
```

1. Properties

- **Spring-client**

```
spring.application.name=client
# client server
server.port=${CLIENT_SERVER_PORT:8080}
server.servlet.context-path=/client
## Cassandra
spring.cassandra.keyspace-name=${CASSANDRA_KEYSPACE}
spring.cassandra.contact-points=${CASSANDRA_HOST}
spring.cassandra.port=${CASSANDRA_PORT}
spring.cassandra.local-datacenter=${CASSANDRA_DATACENTER}
spring.cassandra.username=${CASSANDRA_USERNAME}
spring.cassandra.password=${CASSANDRA_PASSWORD}
spring.cassandra.schema-action=CREATE_IF_NOT_EXISTS
spring.cassandra.connection.init-query-timeout=5000
spring.cassandra.request.timeout=10000
```

```
spring.cassandra.connection.connect-timeout=10000
spring.cassandra.pool.heartbeat-interval=30000
# MySQL
spring.datasource.url=${MYSQL_URL}
spring.datasource.username=${MYSQL_USERNAME}
spring.datasource.password=${MYSQL_PASSWORD}
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# JPA
spring.jpa.show-sql=false
spring.jpa.hibernate.ddl-auto=update
spring.jpa.defer-datasource-initialization=true
spring.jpa.properties.hibernate.show_sql=false
spring.jpa.properties.hibernate.format_sql=false
spring.jpa.properties.hibernate.use_sql_comments=false
# Elasticsearch
spring.elasticsearch.uris=${ELASTICSEARCH_URL}
spring.elasticsearch.username=${ELASTICSEARCH_USERNAME}
spring.elasticsearch.password=${ELASTICSEARCH_PASSWORD}
# JWT
jwt.access-expiration-time=99999999999
jwt.refresh-expiration-time=99999999999
jwt.secret=${JWT_SECRET}
# Firebase
fcm.key.path=${FCM_KEY_PATH}
fcm.key.scope=https://www.googleapis.com/auth/cloud-platform
# Redis
spring.data.redis.host=${REDIS_HOST:localhost}
spring.data.redis.port=${REDIS_PORT:6379}
spring.data.redis.password=${REDIS_PASSWORD}
# MongoDB
spring.data.mongodb.host=${MONGODB_HOST:localhost}
spring.data.mongodb.port=${MONGODB_PORT:27017}
spring.data.mongodb.database=${MONGODB_DATABASE:odd}
spring.data.mongodb.authentication-database=odd
spring.data.mongodb.username=${MONGODB_USERNAME:}
spring.data.mongodb.password=${MONGODB_PASSWORD:}
spring.data.mongodb.auto-index-creation=true
#FastAPI
fastapi.base.url=${FASTAPI_BASE_URL}
```

- **Spring-manager**

```
spring.application.name=manager

# manager server
server.port=${MANAGER_SERVER_PORT:8081}
server.servlet.context-path=/manager
```

```
# jpa
spring.jpa.show-sql=false
spring.jpa.hibernate.ddl-auto=update
spring.jpa.defer-datasource-initialization=true
spring.jpa.properties.hibernate.show_sql=false
spring.jpa.properties.hibernate.format_sql=false
spring.jpa.properties.hibernate.use_sql_comments=false

spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.D

# admin account
admin.account.id=${ADMIN_ACCOUNT_ID}
admin.account.password=${ADMIN_ACCOUNT_PASSWORD}

# baseurl
baseurl=${BASE_URL}
```

2.3. 추천 알고리즘 빌드 방법

1. 버전
    a. Python 3.9
2. 빌드

```
uvicorn app.main:app --reload
```

3. requirements.txt

```
annotated-types==0.7.0
anyio==4.6.2.post1
argon2-cffi==21.3.0
argon2-cffi-bindings==21.2.0
bcrypt==3.2.0
cassandra-driver==3.25.0
cffi==1.15.0
click==8.1.7
colorama==0.4.6
cryptography==37.0.4
exceptiongroup==1.2.2
fastapi==0.115.5
greenlet==3.1.1
h11==0.14.0
idna==3.10
Jinja2==3.0.3
MarkupSafe==2.1.1
mkl-service==2.4.0
mkl_fft>=1.3.0
mkl-random==1.2.2
```

```
numpy==1.24.3
pandas==2.2.3
passlib==1.7.4
pycparser==2.21
pydantic==2.9.2
pydantic_core==2.23.4
PyMySQL==1.1.1
python-dateutil==2.9.0
python-dotenv==1.0.1
pytz==2024.2
six==1.16.0
sniffio==1.3.1
SQLAlchemy==2.0.36
starlette==0.41.2
typing-extensions>=4.8.0
tzdata==2024.2
uvicorn==0.32.0
scikit-learn
```

2.4. 배포하기

1. 배포 환경
    a. AWS EC3 (Ubuntu 20.04.6)
    b. AWS S3
    c. Docker 27.3.1
    d. Jenkins 2.479.1
2. 배포 방법
    a. 설치
        i. Docker 설치

```
# apt update
sudo apt update

# 패키지 설치
$sudo apt-get install apt-transport-https ca-certificates curl gnupg

# gpg 설치 및 도커 저장소 key 저장
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo

# 도커 다운로드 및 레포지토리에 추가
sudo add-apt-repository "deb [arch=amd64] https://download.docker.co

# 도커 설치
```

```
apt-cache policy docker-ce
sudo apt install docker-ce
```

ii. jenkins 설치

```
# port 설정: 9090포트로 외부에서 접속
# volum mount 통해 로컬과 연결
# sock 통해 인증
docker run -d --name jenkins \
    -p 9090:8080 \
    -v /var/jenkins_home:/var/jenkins_home \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /usr/bin/docker:/usr/bin/docker \
    -v /usr/local/bin/docker-compose:/usr/local/bin/docker-compose \
    -e TZ=Asia/Seoul \
    --privileged --user root jenkins/jenkins:lts-jdk17
```

b. CI/CD 환경 구축

i. NginX를 위해 미리 네트워크 생성

```
docker network create runnerway
```

ii. Gitlab과 연결

1. Gitlab에서 access token 발급
2. Jenkins credentials에 access token 등록
    a. Username with password로 생성하여, 본인 Gitlab ID 토큰 입력
3. Jenkinsdp pipe line item 생성
4. Gitlab build trigger 설정
    a. push
5. Gitlab Web Hook 걸기
    a. project hook에서 jenkins project의 url 넣기
    b. token 넣기

iii. Mattermost와 연결

1. Mattermost 채널 생성
2. Incomming Webhook 생성
3. Jenkins에서 Global Mattermost Notifier Settings 설정

iv. NginX 설정

1. /etc/nginx/templates/runnerway.conf.template

```
server {
```

```
    listen 80;
    server_name oodongdan.com;

    return 308 https://oodongdan.com$request_uri;
}

server {
    listen 443 ssl;
    server_name oodongdan.com;

    ssl_certificate /etc/letsencrypt/live/oodongdan.com/fullchain
    ssl_certificate_key /etc/letsencrypt/live/oodongdan.com/privk

    location / {
        proxy_pass http://react-web:80;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_f
        proxy_set_header Host $http_host;

        proxy_buffer_size          128k;
        proxy_buffers              4 256k;
        proxy_busy_buffers_size    256k;
    }

    location /client {
        charset utf-8;
        proxy_pass http://spring-client:8080;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_f
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /manager {
        charset utf-8;
        proxy_pass http://spring-manager:8081;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_f
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /recommend {
        charset utf-8;
        proxy_pass http://fastapi:8000;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_f
```

```
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

1. 443 port로 들어오면 url 바탕으로 proxy
    a. /client가 붙어 있는 경우, spring-client Container의 8080 port로 이동
    b. /manager가 붙어 있는 경우, spring-manager Container의 8081 port로 이동
    c. /recommend가 붙어 있는 경우, fastapi Container의 8000 port로 이동
    d. 그 외 모든 경로, react-web Container의 80 port로 이동
    e. docker 내부 dns 서버를 활용하여 spring-client, spring-manager, fastapi, react-web 컨테이너 찾기
    f. 인증서를 사용하여 https 설정

v. 프론트 CI/CD 구축

1. 파이프라인 구축

```
pipeline {
    agent any

    environment {
        REACT_IMAGE = 'react-web'  // React 애플리케이션을 위한 이미지 이름
        COMPOSE_FILE = 'docker-compose.yml'  // 사용할 Docker Compose 파일
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리 체크아웃
                git branch: 'develop/FE',  // 사용할 브랜치
                url: 'https://lab.ssafy.com/s11-final/S11P31S105.git',
                credentialsId: 'gitlab-access-token'  // Jenkins에 설정된 Credential ID
            }
        }

        stage('Env') {
            steps {
                dir("frontend/web/odd") {
                    sh '''
                        touch .env
                        echo REACT_APP_BASE_URL=[URL] >> .env
                    '''
                }
```

```
            }
        }

        stage('Build Docker Image for React') {
            steps {
                script {
                    dir('frontend/web/odd'){
                        // Docker 이미지 빌드
                        sh 'docker build -t ${REACT_IMAGE} -f d
ockerfile .'
                    }
                }
            }
        }

        stage('Docker Compose Up') {
            steps {
                script {
                    dir('frontend'){
                    // Docker Compose를 사용해 서비스 시작
                        sh 'docker-compose -f ${COMPOSE_FILE} d
own'  // 기존 서비스 중지
                        sh 'docker-compose -f ${COMPOSE_FILE} u
p -d'  // 새 서비스 시작
                        sh 'pwd'
                    }
                }
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    sh 'docker restart nginx'
                }
            }
        }
    }

    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --prett
y=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pre
tty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUIL
```

```
D_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|D
etails>)",
                endpoint: 'https://meeting.ssafy.com/hooks/skwn
d3un5bdsprtj1fsb9hziha',
                channel: 's105_build_result'
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --prett
y=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pre
tty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUIL
D_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|D
etails>)",
                endpoint: 'https://meeting.ssafy.com/hooks/skwn
d3un5bdsprtj1fsb9hziha',
                channel: 's105_build_result'
                )
            }
        }
    }
}
```

1. push,merge가 들어옴
2. 사용할 branch로 checkout
3. env 파일 생성 및 설정
4. doker file을 실행시켜 image 생성

```
# 1단계: React 애플리케이션 빌드
FROM node:20.15.0 AS build

# 작업 디렉토리 설정
WORKDIR /app

# package.json 및 package-lock.json 복사
COPY package*.json ./

# 의존성 설치
RUN npm install

# 애플리케이션 코드 복사
COPY . .
```

```
# 애플리케이션 빌드
RUN npm run build

# 2단계: Nginx로 React 애플리케이션 제공
FROM nginx:alpine

# 빌드된 파일을 Nginx의 HTML 디렉토리로 복사
COPY --from=build /app/build /usr/share/nginx/html

# Nginx 설정 파일을 복사
COPY default.conf /etc/nginx/conf.d/default.conf

# 80번 포트 오픈
EXPOSE 80

# Nginx 서버 시작
CMD ["nginx", "-g", "daemon off;"]
```

1. node:20.15.0 이미지 선택 하여 build환경 설정
2. 디렉토리 선택
3. 사용할 패키지를 담은 파일 복사
4. 의존성 설치
5. react 애플리케이션 빌드
6. nginx를 활용하여 해당 애플리케이션 실행
    a. 기본 제공 html 코드를 복사
    b. 80번 포트 오픈
    c. 해당 포트로 nginx서버 실행

5. compose 파일을 통해서 container들 생성

```
version: '3.8'

services:
  react:
    container_name: react-web
    image: react-web
    environment:
      - TZ=Asia/Seoul
    networks:
      - odd

networks:
  odd:
    external: true
```

1. react 컨테이너 생성
2. network 연결

6. nginx 재시작
7. 빌드 성공/실패 여부 Mattermost 알림

vi. 백엔드 CI/CD 구축

1. Pipe line 구축

```
pipeline {
    agent any

    environment {
        SPRING_IMAGE = 'spring-client' // Spring 백엔드 이미지 이름 
        SPRING_MANAGER_IMAGE = 'spring-manager'
        COMPOSE_FILE = 'docker-compose.yml' // 도커 컴포즈 파일 경로
        // 환경변수 설정
        // ex) MYSQL_URL = 'jdbc:mysql://localhost:3306/odd'
    }
    tools {
        gradle 'gradle' // Gradle 도구 사용
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'develop/BE',
                url: 'https://lab.ssafy.com/s11-final/S11P31S105.
                credentialsId: 'gitlab-access-token'
            }
        }

        stage('Modify Dockerfile') {
            steps {
                script {
                    sh """
                    echo "ENV MYSQL_URL=${MYSQL_URL}" >> backend/
                    echo "ENV MYSQL_USERNAME=${MYSQL_USERNAME}" >
                    echo "ENV MYSQL_PASSWORD=${MYSQL_PASSWORD}" >
                    echo "ENV CLIENT_SERVER_PORT=${CLIENT_SERVER_
                    echo "ENV CASSANDRA_HOST=${CASSANDRA_HOST}" >
                    echo "ENV CASSANDRA_PORT=${CASSANDRA_PORT}" >
                    echo "ENV CASSANDRA_KEYSPACE=${CASSANDRA_KEYS
                    echo "ENV CASSANDRA_USERNAME=${CASSANDRA_USER
                    echo "ENV CASSANDRA_PASSWORD=${CASSANDRA_PASS
                    echo "ENV CASSANDRA_DATACENTER=${CASSANDRA_DA
                    echo "ENV ELASTICSEARCH_URL=${ELASTICSEARCH_U
                    echo "ENV ELASTICSEARCH_USERNAME=${ELASTICSEA
```

```
                    echo "ENV ELASTICSEARCH_PASSWORD=${ELASTICSEA
                    echo "ENV JWT_SECRET=${JWT_SECRET}" >> backen
                    echo "ENV FCM_KEY_PATH=${FCM_KEY_PATH}" >> ba
                    echo "ENV REDIS_HOST=${REDIS_HOST}" >> backen
                    echo "ENV REDIS_PORT=${REDIS_PORT}" >> backen
                    echo "ENV REDIS_PASSWORD=${REDIS_PASSWORD}" >
                    echo "ENV MONGODB_HOST=${MONGODB_HOST}" >> ba
                    echo "ENV MONGODB_PORT=${MONGODB_PORT}" >> ba
                    echo "ENV MONGODB_DATABASE=${MONGODB_DATABASE
                    echo "ENV MONGODB_USERNAME=${MONGODB_USERNAME
                    echo "ENV MONGODB_PASSWORD=${MONGODB_PASSWORD
                    echo "ENV FASTAPI_BASE_URL=${FASTAPI_BASE_URL
                    echo "ENV ADMIN_ACCOUNT_ID=${ADMIN_ACCOUNT_ID
                    echo "ENV ADMIN_ACCOUNT_PASSWORD=${ADMIN_ACCO
                    echo "ENV BASE_URL=${BASE_URL}" >> backend/ma
                    """
                }
            }
        }

        stage('application-key download') {
            steps {
                dir('backend/client') {
                    withCredentials([file(credentialsId: 'applica
                        script {
                            sh 'cp $applicationKey src/main/resou
                            sh 'ls src/main/resources'
                            sh 'cat src/main/resources/applicatio
                        }
                    }
                    withCredentials([file(credentialsId: 'fcmmsg'
                        script {
                            sh 'cp $fcmMsg src/main/resources/fcm
                            sh 'ls src/main/resources'
                            sh 'cat src/main/resources/fcmmsg.jso
                        }
                    }
                }
            }
        }

        stage('Build Spring Boot') {
            steps {
                script {
                    // client 빌드
                    dir('backend/client') {
                        sh 'chmod +x gradlew'
                        sh './gradlew build -x test'
```

```
                }
                // manager 빌드
                dir('backend/manager') {
                    sh 'chmod +x gradlew'
                    sh './gradlew build -x test'
                }
            }
        }
    }


    stage('Stop and Remove Existing Containers') {
        steps {
            script {
                dir('backend') {
                    sh 'docker-compose -f ${COMPOSE_FILE} dow
                }
            }
        }
    }

    stage('Remove Existing Docker images') {
        steps {
            script {
                dir('backend') {
                    sh "docker rmi -f \$(docker images -q ${S
                    sh "docker rmi -f \$(docker images -q ${S
                }
            }
        }
    }

    stage('Build Docker Images Spring') {
        steps {
            script {
                // client 이미지 빌드
                dir('backend/client') {
                    sh 'docker build -t ${SPRING_IMAGE} .'
                }
                // manager 이미지 빌드
                dir('backend/manager') {
                    sh 'docker build -t ${SPRING_MANAGER_IMAG
                }
            }
        }
    }
```

```
        stage('docker-compose up') {
            steps {
                script {
                    dir('backend') {
                        sh """
                        CLIENT_SERVER_PORT=${CLIENT_SERVER_PORT}
                        CASSANDRA_PORT=${CASSANDRA_PORT} CASSANDR
                        CASSANDRA_USERNAME=${CASSANDRA_USERNAME}
                        MYSQL_URL=${MYSQL_URL} MYSQL_USERNAME=${M
                        ELASTICSEARCH_URL=${ELASTICSEARCH_URL} EL
                        ELASTICSEARCH_PASSWORD="${ELASTICSEARCH_P
                        FCM_KEY_PATH="${FCM_KEY_PATH}" REDIS_HOST
                        docker-compose -f ${COMPOSE_FILE} up -d
                        """
                    }
                }
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    sh 'docker restart nginx'
                }
            }
        }
    }

    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=
                def Author_Name = sh(script: "git show -s --prett
                mattermostSend(color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_N
                endpoint: 'https://meeting.ssafy.com/hooks/skwnd3
                channel: 's105_build_result')
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=
                def Author_Name = sh(script: "git show -s --prett
                mattermostSend(color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_N
                endpoint: 'https://meeting.ssafy.com/hooks/skwnd3
                channel: 's105_build_result')
            }
```

```
        }
    }
}
```

1. push, merge 액션 발생
2. branch 이동
3. dockerfile에 환경변수 저장 및 seceret config 파일 download
    a. Jenkins Credentials에 등록된 applicaion-key.yml
    b. Jenkins Credentials에 등록된 fcmmsg.json
4. build
5. 생성된 build 파일의 jar 파일 image화
    a. docker file을 미리 설정
    b. docker file 코드

```
FROM openjdk:17-alpine
RUN apk add --no-cache tzdata
ARG JAR_FILE=/build/libs/runnerway-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

6. 해당 image를 Build ⇒ 이미지 생성
7. docker-compose를 내리고, 생성된 image를 다시 실행

```
version: '3.8'

services:
  spring-client:
    container_name: spring-client
    image: spring-client
    environment:
      - TZ=Asia/Seoul
    privileged: true
    networks:
      - odd

  spring-manager:
    container_name: spring-manager
    image: spring-manager
    environment:
      - TZ=Asia/Seoul
    privileged: true
    networks:
      - odd
```

```
networks:
  odd:
    external: true
```

1. image 실행하여 container 생성
2. network 포함하여 연결, 해당 container에 접근 가능하도록 함

8. NginX 재실행
9. 빌드 성공/실패 여부 Mattermost 알림

vii. 추천 알고리즘 CI/CD 구축

1. Pipe line 구축

```
pipeline {
    agent any

    environment {
        PYTHON_IMAGE = 'fastapi' // fastapi 이미지 이름 설정
        COMPOSE_FILE = 'docker-compose.yml' // 도커 컴포즈 파일 경로
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리에서 'develop/BE' 브랜치를 체크아웃
                git branch: 'develop/PY',
                url: 'https://lab.ssafy.com/s11-final/S11P31S105.
                credentialsId: 'gitlab-access-token' // GitLab 인
            }
        }

        stage('.env download') {
            steps {
                dir('fastapi') {
                    withCredentials([file(credentialsId: '.env',
                        script {
                            // .env 파일을 lightfm-api 디렉토리로 복사
                            sh 'cp $envFile .env'
                            // 확인을 위해 .env 파일의 내용을 출력
                            sh 'ls -la'
                            sh 'cat .env'
                        }
                    }
                }
            }
        }

        stage('Build Docker Image fastapi') {
```

```
            steps {
                dir('fastapi') {
                    script {
                        sh 'docker build --no-cache -t ${PYTHON_I
                    }
                }
            }
        }

        stage('Stop and Remove Existing Containers') {
            steps {
                dir('fastapi') {
                    script {
                        sh 'docker-compose -f ${COMPOSE_FILE} dow
                    }
                }
            }
        }

        stage('Remove Existing Docker images') {
            steps {
                script {
                    sh "docker rmi -f \$(docker images -q ${PYTHO
                }
            }
        }

        stage('docker-compose up') {
            steps {
                dir('fastapi') {
                    script {
                        sh 'docker-compose -f ${COMPOSE_FILE} up
                    }
                }
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    // Nginx 컨테이너 재시작
                    sh 'docker restart nginx'
                }
            }
        }
    }

    post {
```

```
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=
                def Author_Name = sh(script: "git show -s --prett
                mattermostSend (color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_N
                endpoint: 'https://meeting.ssafy.com/hooks/skwnd3
                channel: 's105_build_result'
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=
                def Author_Name = sh(script: "git show -s --prett
                mattermostSend (color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_N
                endpoint: 'https://meeting.ssafy.com/hooks/skwnd3
                channel: 's105_build_result'
                )
            }
        }
    }
}
```

1. push, merge 액션 발생
2. branch 이동
3. .env 파일 download
    a. Jenkins Credentials에 등록된 .env
4. build
5. docker file 통해 image 생성
    a. docker file 코드

```
# Python 3.9 이미지 사용
FROM python:3.9

# 작업 디렉토리 설정
WORKDIR /app

# requirements.txt 파일 복사
COPY requirements.txt .

# 의존성 설치
RUN pip install --no-cache-dir --upgrade -r requirements.tx

# 필요한 시스템 패키지 설치
```

```
RUN apt-get update && apt-get install -y libgl1-mesa-glx &&

# 애플리케이션 소스 코드 복사
COPY . .

# FastAPI 서버 실행
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--por
```

1. python:3.9 이미지 선택
2. 디렉토리 설정
3. requirements.txt 통해 의존성 설치
4. 시스템 패키지 설치
5. FastAPI 서버 실행
    a. 8000번 포트 오픈

6. docker-compse 통해 container 생성

```
version: '3.8'

services:
  fastapi:
    container_name: fastapi
    image: fastapi
    build:
      context: .
      dockerfile: Dockerfile
    env_file:
      - .env
    environment:
      - TZ=Asia/Seoul
    ports:
      - "8000:8000"
    networks:
      - odd

networks:
  odd:
    external: true
```

1. fastapi container 생성
2. network 연결

7. NginX 재실행
8. 빌드 성공/실패 여부 mattermost 알림

c. 포트 개방

- SSAFY EC2(t2.xlarge)

1. 80 ⇒ http로 접속을 위해 개방
2. 3306 ⇒ MySQL
3. 6379 ⇒ Redis
4. 8080 ⇒ 백엔드 Spring-client 통신
5. 8081 ⇒ 백엔드 Spring-manager 통신
6. 9090 ⇒ Jenkins 접속
7. 9200 ⇒ Elasticsearch
8. 27017 ⇒ MongoDB

- EC2 인스턴스 1(t3.2xlarge)
  1. 80 ⇒ http로 접속을 위해 개방
  2. 8000 ⇒ FastAPI 통신
  3. 9090 ⇒ Jenkins 접속
  4. 7077, 8080 ⇒ Spark Master
  5. 8081 ⇒ Spark Worker
  6. 9042, 9043, 9044 ⇒ Cassandra
- EC2 인스턴스 2(t2.xlarge)
  1. 7077, 8080 ⇒ Spark Master
  2. 8081 ⇒ Spark Worker
  3. 27017 ⇒ MongoDB

3. 프로젝트에서 사용하는 외부 서비스 정보를 정리한 문서

3.1. Firebase

- 사용 목적: 알림 기능
- 환경 변수 설정:
  - FrontEnd

    app(frontend/app/odd/android/build.gradle)

    ```
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_ve
        classpath 'com.android.tools.build:gradle:8.1.0' // 최신 버전으로
        classpath 'com.google.gms:google-services:4.3.14'
    }
    ```

  - BackEnd

    Spring-client(backend/client/src/main/resources/application.properties)

    ```
    # Firebase
    fcm.key.path=${FCM_KEY_PATH}
    ```

```
fcm.key.scope=https://www.googleapis.com/auth/cloud-platform
```

3.2. S3 Bucket

- 사용 목적 : 사진, 파일 저장
- 환경 변수 설정

```
S3_NAME=****
S3_REGION=****
S3_ACCESS_KEY_ID=****
S3_SECRET_ACCESS_KEY=****
```

- 필요 정보
  - AWS 계정 생성
  - S3 Bucket 생성
  - 접근 키 및 비밀 키 발급

4. DB 덤프 파일 최신본(첨부 파일)

5. 시연 시나리오(첨부 파일)